Effective Decision Support for Product Configuration by Using CBR

Hiroya Inakoshi, Seishi Okamoto, Yuiko Ohta and Nobuhiro Yugami
FUJITSU LABORATORIES LTD.,
9-3, Nakase 1-chome, Mihama-ku, Chiba City, Chiba 261-8588, Japan
{inakoshi,seishi,yuiko,yugami}@flab.fujitsu.co.jp

Abstract

In this paper, we propose a new framework for product configuration that integrates a constraint satisfaction problem with case-based reasoning (CBR), and this framework is applied to an on-line sales system. Given a user query, CBR first retrieves similar cases from the case base in which past successful configurations are stored. Then, it formalizes desirability criteria of products in accordance with the user query by using the similar cases. Both the user query and desirability criteria, as well as the definition of a product family that is obtained in advance, composes a configuration model to be processed with a constraint-based configurator. This configurator processes this configuration model as a constraint satisfaction problem with ranking its solutions by the desirability criteria. Thus, the solutions that the integrated framework provides can be appropriate to users and simultaneously correct in the definition of product family. We conducted several examinations to evaluate the performance of our on-line sales system. The results of these examinations show that our framework surely enlarges opportunities when users of our system find useful configurations within a few configurations, even if they specify a few query items.

Introduction

Many enterprises have manufactured products that satisfy a significant number of people's needs. According to marketing research results, these products have been designed to suit a certain category of people. However, enterprises are moving away from the one-size-fits-all products to customizable families of products. Since products in these product families can be arranged to suit individuals, these products can attract the interest of many people, and the satisfaction of people who acquire these products is likely to increase. The product families thereby bring strength to enterprises in the marketplace, and as a result, product configuration has become a hot topic of investigation (Sabin & Weigel 1998); for example, it has been applied to the design of computer systems (Barker & O'Connor 1989) and telephone switching systems (Fleischanderl *et al.* 1998). Electronic commerce is appealing to many people as well as enterprises. Product configuration has a part in electronic commerce, since complex products and services, such as insurance, travel, and personal computers, are expected to be arranged for individuals via the internet.

Product configuration is a problem to present the most satisfactory configurations for user requirement while observing the definition of product family. A product family, including the products handled in electronic commerce, consists of generic products and a variety of parts implementing their particular functions. Though products in a product family are designed so that they can be set up in any combination with their standardized interfaces, there are restrictions on their connections to one another. Other restrictions also have to be satisfied in product assembly, and descriptions of desirability about what combination is good have to be created according to specific purposes and situations. Each configuration methodology has a particular model that represents configuration tasks, and these tasks are carried out with the process control strategy of the methodology. Product configuration thus makes it possible to arrange products quickly and avoid use of the wrong configurations.

However, a large part of complexity of product configuration lies in this representing of these models so that they should have expressiveness and representational power. An entire configuration model, including not only a definition of product family but also a description of desirability that users may have for products individually, is required to provide accurate solutions. In addition, configuration methodologies should have mechanisms to cope with the frequent change of their models. It is because enterprises change definitions of product families within very short periods of time since newer and better products appeal to more people. Every time a product family changes, the corresponding configuration model must be modified accordingly. This acquisition and maintenance of configuration models is critical for any configuration methodology, and it requires much effort and expertise as well as costs.

There are many approaches to product configuration (Barker & O'Connor 1989; Mittal & Fraymann 1989; Stumptner & Wotawa 1998), including constraint-based and case-based approaches, both of which are related to this work.

A constraint-based approach (Fleischanderl *et al.* 1998; Gelle & Weigel 1996) achieve generic and domain-independent process control of product configuration. That is, once a configuration model is obtained in the form of variables and constraints, it can be processed with a generic methodology, which is the constraint satisfaction problem (CSP) technique. In such an approach, the variables represent the functions, properties, and parts of a product family, and the constraints on the variables define their relationship and restrictions on connectivity. However, users have to adequately and precisely express their demands concerning the products if they expect the desired solutions. Moreover, if a user makes an incorrect demand in the definition of a product family, configuration systems cannot report any solutions. In this event, the task of the configuration system is to detect the incorrect item and report it to the user. These situations are not convenient for users, especially for ordinary users in electronic commerce because they are users without expertise who do not have concrete knowledge about product families.

A case-based approach (Rahmer & Voss 1996; Stahl & Bergmann 2000) can work without a complete configuration model. Instead, such an approach uses configurations that are similar to user requirements as solutions. Thus, a case-based approach greatly decreases the required effort to obtain and maintain configuration models. However, a case-based approach sometimes requires a case adaptation phase in which the similar configurations must be modified to suit the current requirements. It is thus necessary to acquire the requisite knowledge and keep up-to-date on the relevant information to effect a case adaptation. This is another problem of model acquisition.

We propose a new framework of product configuration that integrates CSP with CBR to provide desired products by obtaining the configuration models tailored to individual user queries. And we applied this integrated framework to an on-line sales system. In this framework, CBR fist generates desirability criteria dynamically, estimating the desirability of products for a user by using the similar cases to the given query. Then, a constraint-based configurator ranks resultant configurations by the desirability criteria, while these configurations should strictly observe both the user query and the definition of product family. Even if the user query is incorrect in the definition of product family, this integrated framework presents alternative products by modifying the user query.

Moreover, our system possesses a navigation function to quickly direct users to the products they want by outputting good query items as their next selection. Although this navigation function is not a part of the configuration process, it provides decision support to users.

Integrated Configuration Framework

Figure 1 shows a new framework for product configuration that integrates CSP with CBR. Above the constraint satisfaction problem solver, which is named CSP solver, the CBR part in our framework serves as a wrapper for the problem solver. For this reason, we call this CBR part "*CBR Wrapper*." The components in our framework are described below:

Case base: The case base consists of the past cases, each of which pairs a user query and its corresponding configuration. The case base may therefore contain repetition of the same or very similar queries and configurations, if the records represent typical cases in the marketplace.

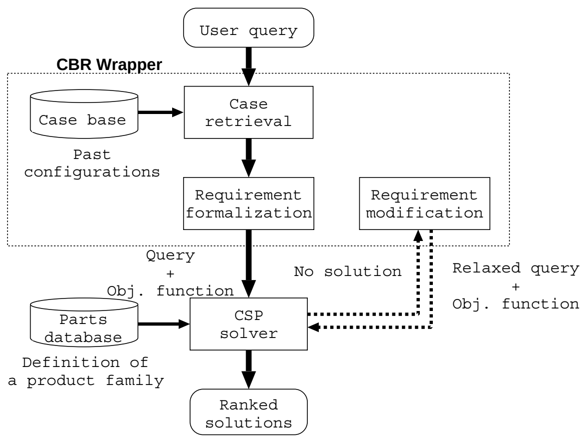


Figure 1: Framework for product configuration.

Case retrieval: Similar cases are retrieved from the case base in accordance with the similarities between the current query and the cases. The similar cases are used to formalize user requirements so that they are well-defined. They are not used as direct solutions.

Requirement formalization: An object function is dynamically generated by using similar cases to the current query. A well-defined requirement consists of the current query and the object function, and it is supplied to CSP solver.

Requirement modification: The well-defined requirement is modified only if there is no configuration and CSP solver returns no solution back to CBR Wrapper. The modified well-defined requirement is then supplied to CSP solver again.

Parts database: A parts database contains the definition of a product family. It defines the types of parts, the constraints on parts connectivity, and other kinds of restrictions on the products.

CSP solver: CSP solver receives a well-defined user requirement and solves the problem specified by three objects, which are the current query, the object function, and the definition of product family in a parts database. CSP solver provides a solution to users. If no configuration is possible for the current problem, CSP solver returns no solution back to CBR Wrapper.

A user's explicit demand for products is represented as a user query. The user query consists of conditions on parts, functions, and properties of the products. CBR Wrapper formalizes the user's demand so that it is well-defined, and the formalization consists of the user query and the object function generated with similar cases to the query. The definition of the relevant product family is already retained in a parts database. However, the entire configuration model must include a kind of description of desirability as well. Acquisition of this desirability is a delicate operation in general and requires expertise about the target domain. CBR Wrapper obtains this description of desirability as the object function so as to be processed with CSP solver, estimating the desirability of products for a user, even from a few conditions supplied using similar cases to the query.

CSP solver solves the constraint satisfaction problem of the configuration model with ranking its solutions in accordance with the description of desirability; CSP solver provides the solutions that simultaneously satisfy all of the constraints and restrictions in a part database as well as the current query, and these solutions are ranked by the object function at the same time. As a result, users can find the most useful configurations.

Occasionally, there is no configuration that satisfies the user query. This is because the current query is incorrect in the definition of a product family. Users may supply such an incorrect query, especially ordinary users in electronic commerce, because they do not know much about the product family. In this event, CBR Wrapper modifies the original query, and CSP solver works on another CSP task with the modified query. Thus, users obtain alternative configurations and may have an opportunity to receive useful configurations.

Application to E-Commerce

We applied this integrated framework to an on-line sales system for personal computers. Personal computers are the target product configuration in which we are interested, because they are made up of many parts, including CPU, memory cards, a CD-R/RW, and other peripheral devices. These parts must be arranged correctly in the definition of a product family as well as by usefulness according to a user's needs.

The first part of this section is a description of the system and the user interfaces of our system. That is followed by a description of the problem and explanation of the concrete implementation of our kernel system. The final part is about *navigation*, which is a kind of decision guide. Although this is not a part of the configuration, it assists users to find the useful configurations quickly by reporting effective query items according to their purpose.

System

Our system architecture is shown in Figure 2. This system was implemented with the Java^{TM}2 and JavaServer Pages^{TM} (JSP) technologies. JSP is a framework for dynamic generation of HTML pages. Users can submit their queries by filling in a form on an HTML page generated by JSP. The user query is submitted via an HTTP connection and received by our configuration engine. The configuration engine is composed of CBR Wrapper and CSP solver, and it was implemented as a JavaBeans component. JavaBean is an organized object that is automatically instant by Java Virtual Machine to provide a certain set of functions. The configurator bean is active during a session managed by a cookie so that this system can handle multiple accesses from different users concurrently. The user query is parsed and supplied to the configurator bean. Then, the bean performs the configuration task and returns a solution. JSP generates an HTML page that reports the solution and send it back via an HTTP connection. In addition, the CBR Wrapper class implements navigation as one of its member functions. The results of navigation are also reported on HTML pages that are also returned via HTTP connections.

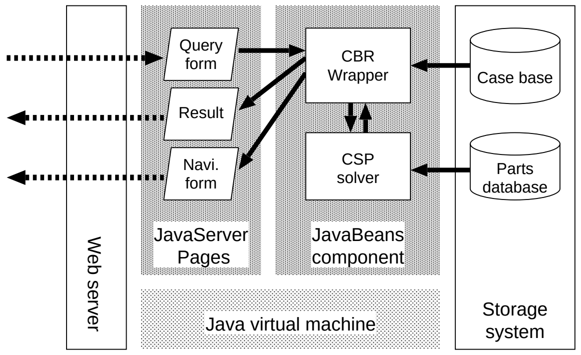
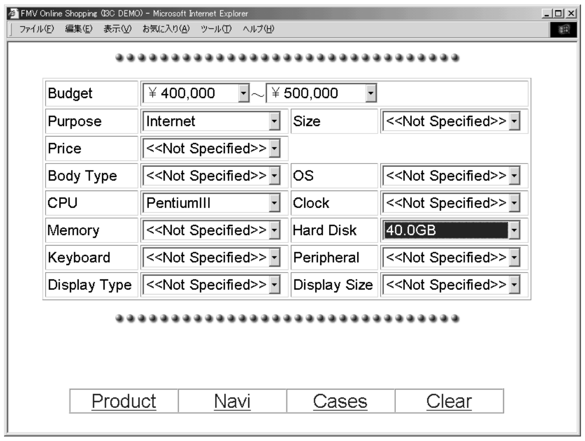


Figure 2: Physical architecture of our configuration system.

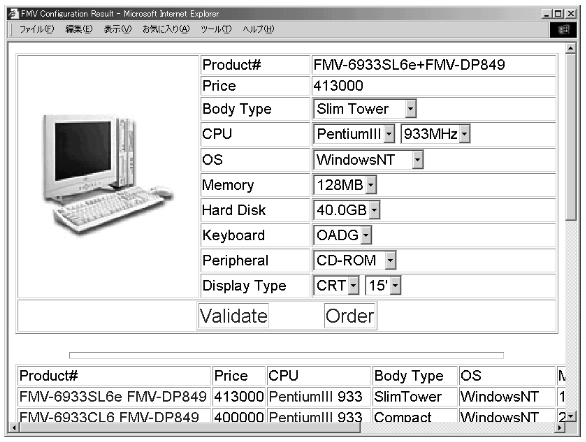
Figure 3 shows the query form of our online shopping system. Users interact with our system with this query form. Users can specify their queries by selecting items displayed on this form. Clicking the "Submit" button displays the resulting configurations in a results window. Clicking the "Navi." button highlights an effective query item for the next configuration.


FMV Online Shopping (I3C DEMO) - Microsoft Internet Explorer
ファイル(F) 編集(E) 表示(V) お気に入り(A) ツール(T) ヘルプ(H)
Product
Navi
Cases
Clear

Budget	￥400,000 ～ ￥500,000		
Purpose	Internet	Size	<<Not Specified>>
Price	<<Not Specified>>		
Body Type	<<Not Specified>>	OS	<<Not Specified>>
CPU	PentiumIII	Clock	<<Not Specified>>
Memory	<<Not Specified>>	Hard Disk	40.0GB
Keyboard	<<Not Specified>>	Peripheral	<<Not Specified>>
Display Type	<<Not Specified>>	Display Size	<<Not Specified>>

Figure 3: Query form of our online shopping system.

Figure 4 is the results window for a query submitted with the query form. At the top of the window, an image of the product ranked at the top is displayed. Users can further customize the product by selecting the items to the right of the image, and the customized product is validated by clicking the "Check" button. Satisfactory configurations and, sometimes, alternative configurations are displayed at the bottom of the window.


FMV Configuration Result - Microsoft Internet Explorer
ファイル(F) 編集(E) 表示(V) お気に入り(A) ツール(T) ヘルプ(H)
Validate
Order

Product#	FMV-6933SL6e+FMV-DP849
Price	413000
Body Type	Slim Tower
CPU	PentiumIII 933MHz
OS	WindowsNT
Memory	128MB
Hard Disk	40.0GB
Keyboard	OADG
Peripheral	CD-ROM
Display Type	CRT 15'

Product#	Price	CPU	Body Type	OS	M
FMV-6933SL6e FMV-DP849	413000	PentiumIII 933	SlimTower	WindowsNT	1
FMV-6933CL6 FMV-DP849	400000	PentiumIII 933	Compact	WindowsNT	2

Figure 4: Results window that opens when "Submit" button on query form is clicked.

Table 1: All features in our system and their number of values.

Feature	#values	note	Feature	#values	note
`Budget`	–	Numerical	`Hard Disk`	2	
`Purpose`	2	Utility-based	`Keyboard`	2	
`Body Type`	2		`Peripheral`	4	
`OS`	2		`Display Type`	2	
`CPU`	3		`Display Size`	3	
`Clock`	5		`Product#`	10	Output
`Memory`	4		`Display#`	5	Output

Problem and Procedures

The input to our system is user queries that users submit with the query form shown in Figure 3, and system output is a set of configurations displayed for users in the results window shown in Figure 4. Our system also utilizes cases, which are past configurations. A user query, a configuration, and a case, are represented on the same vector form:

$$\begin{aligned} \text{Query} &= [\texttt{PenIII}, \texttt{Compact}, ?, \cdots, ?, ?], & (1)\\ \text{Product} &= [\texttt{PenIII}, \texttt{Compact}, \texttt{128MB}, \\ &\quad \cdots, \texttt{FMV6800CL6}, \texttt{FMVDP849}], & (2)\\ \text{Case} &= [\texttt{Celeron}, \texttt{SlimTower}, \texttt{128MB}, \\ &\quad \cdots, \texttt{FMV6667SL6c}, \texttt{FMVDP849}]. & (3) \end{aligned}$$

Unknown values, which are represented by '`?`', may occur in these vectors. Our system currently defines the 14 features as shown in Table 1.

With the exception of `Budget`, which takes a numerical value representing price of a personal computer, these features have their own domain and take symbolic values.

12 features out of the 14 features in Table 1 can be specified in a user query. A user query commonly contains many unknown values when a user specifies only a few items. The remaining two features, which are `Product#` and `Display#`, cannot be specified, and they therefore always take unknown values in a user query. The goal of our system is to determine all of the values in output vectors. These output vectors represent the resultant configurations that should simultaneously be appropriate to the current user and correct in the definition of a product family, and they appear in the results window of Figure 4.

Case base Cases in a case base are about the past configurations that our system has provided to users. These cases are represented in the vector form, and there is no unknown value in them, since the output of our system is vectors without unknown values as we have already explained above. Each feature corresponds to a part or a property of a part, which are the features of a personal computer with the exception of `Purpose`.

The current system has a single *utility-based condition*, `Purpose`, as shown in Table 1. Purpose for buying a personal computer is actually neither a function nor a property of products. This is just utility of a personal computer. For ordinary users in electronic commerce, it is not always easy to express demands in terms of specification level. Utility-based conditions are therefore useful, especially for beginners, for expressing desirability of products.

CBR is strongly suited for such types of reasoning where the model of a problem is hardly exact, such as the one with this *utility*. By only defining utility-based conditions as separate features of a case, our system retrieves similar cases to these utility-based conditions. `Purpose` is retained as a part of a case, if it is supplied by users.

Case retrieval First in configuration processes, similar cases to a user query are retrieved from the case base. Let the user query be q whose feature values are a_j, and case be c whose feature values are b_j. The number of features is m for both the query and case:

$$q = [a_1, a_2, ..., a_m], \quad c = [b_1, b_2, ..., b_m] \ . \tag{4}$$

The distance between the query and case is a sum of local distances of the features whose values are not unknown in the query:

$$d(q, c) = \sum_{j=1}^{m} \delta(a_j, b_j) \ , \tag{5}$$

where $\delta(a_j, b_j)$ is the local distance between the values of feature j of the query and case. Our system determines the local distance between the values of each feature by using the modified value difference metric (MVDM) (Cost & Salzberg 1993) in the case base. Let D be the domain of a target class:

$$\delta(v_{jk}, v_{jl}) = \sum_{i \in D} \left| \frac{N_{ki}}{N_k} - \frac{N_{li}}{N_l} \right|^r \tag{6}$$

where N_{ki} denotes the number of times v_{jk} is classified into category i, and N_k is the total number of times v_{jk} occurred. We defined r as 1 and used feature `Product#` as the target class.

We defined similar cases as the cases whose distances to the query are less than threshold value ε (Okamoto & Yugami 1997): with a threshold value, ε, if $d(q, c) \le \varepsilon$, c is defined as a similar case of q. However, in case that there were a very few similar cases, we employed another definition of similar cases to retrieve a sufficient number of cases in generating an object function: the threshold value, ε, is extended until the number of cases where $d(q,c) \leq \varepsilon$ hold exceeds a certain fixed number. Such cases are defined as the similar cases of q.

Requirement formalization CBR Wrapper dynamically generates an object function by using the current similar cases. Let x be a representation of a configuration, and m be the number of features. $|D_j|$ denotes the cardinality of D_j:

$$x = [u_1, u_2, ..., u_m], \quad v_{jk} \in D_j \ (k = 1, ..., |D_j|) \ . \quad (7)$$

Let F be the object function. For the configuration, x, the value of the object function is calculated as follows, using the probability distribution of each feature in the similar cases:

$$F(x) = \sum_{j=1}^{m} \sum_{v_{jk} \in D_j} \text{Prob}(u_j = v_{jk}) I(u_j = v_{jk}), \quad (8)$$

$$I(u_j = v_{jk}) = \begin{cases} 1 & \text{when } u_j = v_{jk} \\ 0 & \text{otherwise} \end{cases} .$$

Mentioning utility-based conditions, they affect the object function as well as similar cases. Thus, the object function translates the utility-based conditions into the desirability of parts, functions, and properties of products, and this translation is processed with CSP.

Parts database A parts database has a set of binary constraints on two different features. An example of the binary constraint on `Body Type` and `Memory` is shown below:

$$\begin{aligned} &(\texttt{BodyType}, \texttt{Memory}) && (9)\\ &\quad \in \{(\texttt{Compact}, \texttt{64MB}), (\texttt{Compact}, \texttt{128MB}),\\ &\qquad (\texttt{Desktop}, \texttt{128MB}), (\texttt{Desktop}, \texttt{256MB}), ...\} \end{aligned}$$

These constraints are represented as sets of the feature values that a configuration can have at the same time. We defined constraints for 8 pairs of features.

CSP solver Supplied with the object function, F, and the user query, q, CSP solver works on the constraint satisfaction problem defined by the current query, the object function, and constraints on features. CSP solver currently employs a tree-search method enhanced by a lookahead strategy. The tree-search method, basically, assigns in sequence a feature value to each feature with an unknown value in the current query. However, it is time consuming to backtrack and to assign another feature value when the currently assigned feature value violates the constraints. A Lookahead strategy can prevent this backtracking. Before assigning a feature value to the next feature, it reduces the remaining search space by examining the arc-consistency (Tsang 1993) of the search space. Thus, the CSP solver operation becomes backtrack-free, enabling its search method to become efficient.

Requirement modification When there is no configuration for a user query, the current system relaxes the query items of an incompatible query by simply replacing a query item with an unknown value. This relaxation occurs one after another for each feature that a user supplies with a certain value. For example, for the original query, q, a set of relaxed queries is given as Q:

$$\begin{aligned} q &= [\texttt{PenIII}, \texttt{256MB}, \texttt{Note}, ?, \cdots, ?], && (10)\\ Q &= \{[?, \texttt{256MB}, \texttt{Note}, ?, \cdots, ?],\\ &\quad [\texttt{PenIII}, ?, \texttt{Note}, ?, \cdots, ?],\\ &\quad [\texttt{PenIII}, \texttt{256MB}, ?, ?, \cdots, ?]\}. && (11) \end{aligned}$$

The current system relaxes a single query item at one time because the configurations far different from the original and incompatible queries may not be helpful as alternative configurations.

CBR Wrapper provides a relaxed query in Q and the object function, F. CSP solver performs the configuration task with them and returns configurations. This process is iterated for each relaxed query. All of these configurations are aggregated and ranked by object function. Modified configurations that are desirable are ranked higher in the aggregated configurations, although neither CBR Wrapper nor CSP solver performs difficult resolution of incompatibility.

Navigation

Navigation is a kind of interactive decision guide that directs users quickly to their desired configurations after they fill in a few query items. Although navigation is not a function of product configuration itself, it provides useful decision support to users in product configuration. Since our configuration framework employs CBR, it can provide this navigation by utilizing the similar cases used to generate the object function.

There are many properties and functions with which users could express their demands, but users are reluctant to answer a large portion of these questions. By using the current similar cases, our system dynamically determines the most effective query item where the query item is the best at clarifying the desirability of products to users for all the query items not answered so far. Thus, a requirement for a lot of input by users is reduced by presenting the most effective query items to users. Furthermore, it is very helpful if users can express their demands interactively and can specify the most effective query item in every interaction stage. Users may not be expected to actually specify most of the query items and express their demands at one time.

We employ information gain criteria (Quinlan 1993), which is used in decision tree induction, to evaluate the discriminating power of features between cases against class labels. The information gain criteria therefore operates on data with class labels. However, the decision guides have to operate on data without class labels, which appears very likely in actual application, and our case base is also without class labels. To address this lack of class labels, another investigation on decision guides (Doyle & Cunningham 2000) employed clustering to import class labels into data without class labels. Then they calculated information gain values against these class labels.

Our current system employs the following method that handles the lack of class labels differently: each output feature is regarded as the target class, which is a virtual target class, and information gain values are evaluated for all input features that have not been answered so far. We denoted the information gain value of feature j for virtual class t as $info_j(t)$. This is iterated for each virtual target class, and these information gain values are summed for all the virtual target classes:

$$info_j = \sum_{t \neq j} info_j(t)\delta_t \quad , \tag{12}$$

$$\delta_t = \begin{cases} 1 & \text{when } a_t = ? \\ 0 & \text{otherwise} \end{cases} \quad . \tag{13}$$

Feature e, which maximizes $info_j$, is presented to users as the most effective query item for clarifying their desirability of products:

$$e = \arg\max_j \; info_j \quad . \tag{14}$$

Empirical Evaluations

We examined whether our system has the effect that we explained. We should have conducted an evaluation with human subjects, since it is the best way and most practical. However, we started with an empirical evaluation method that simulates the performance of a human user because a sufficient number of human subjects was not available.

We used leave-one-out cross validation as the evaluation procedure: for each case taken out of the case base, our system is examined to determine if it returns *correct* products for the case. Then, the case is put back into the case base for examination with the next case. We defined the correctness of the solutions as follows:

Definition 1 *ε-approximation: Given a case, c, and a solution, x, both of which have no feature with an unknown value, x is defined as being within ε-approximation of c, when the number of features whose values differ between c and x is less than or equal to ε.*

Definition 2 *γ-ε correct: Given a case, c, and ranked solutions, X, X is defined as being γ-ε correct for c when there is a solution, x, that is within ε-approximation of c in the top γ of X.*

We selected *accuracy* to evaluate our system. *Accuracy* means whether the solution contains a suitable product for a query. The accuracy of the leave-one-out cross validation task is defined as a ratio of the number of γ-ε *correct* tests to the number of all cases.

We carried out three examinations to evaluate the effect of our three support functions: ranking configurations, presenting alternative configurations, and navigation. These examinations were conducted on the case base used in the personal computer sales that we have explained thus far. The detail description of this case base is shown in Table 1; in short, there are 14 features all of which take symbolic values with the exception of one numerical feature, `Budget`. We left `Budget` out of consideration because its value can be determined by the other feature values and is therefore irrelevant to the examinations. Given several query items, values of all the features should be determined by our system, and these values are the subject of the test of γ-ε *correctness*. `Purpose` is not a subject of this test because it is only for input and therefore irrelevant to the correctness of the resultant configuration.

```
boolean
examine(c, n, init_query(),  next_query())
  {
    q = init_query(c);
    for (k = 2; k <= n; k++)
      {
        X = configure(q);
        if (is_correct(X,c))
          return true;
        q = next_query(q,c);
      }
    return false;
  }
```

Figure 5: Procedure for examining correctness for a case.

Figure 5 shows the test procedure for every case. We denoted a configuration task for a query, `q`, as `configure(q)`, which returns ranked products, `X`. `is_correct(X,c)` returns true if `X` is γ-ε *correct* for the case, `c`, and false otherwise. The examination procedure simulates human activity, where very few features are specified initially, and then a feature value is interactively supplied until the number of the specified features exceeds the fixed number, `n`. `init_query()` and `next_query()` simulate this interactive human activity. They initialize and update, respectively, a query. Since these initialization and update functions should be different for each purpose of the three examinations, their implementations are explained in separate subsections below.

Evaluation of Ranking Method

In this examination of our ranking method, the initialization and update functions are implemented as follows: the initialization function, `init_query(c)`, first instantiates a query, `q`, by copying the values of the current test case, `c`. Then, it replaces all feature values with unknown values except for two feature values that are left untouched. The two features are determined at random. The update function, `next_query(q,c)`, specifies a single unknown-value feature of the query, `q`, with the corresponding value of the current test case, `c`. The unknown-value feature to be specified is determined using the navigation functions.

Figure 6 shows that our system achieves high accuracy even when very few features are specified. And it also shows that the accuracy goes up drastically with small γ values and becomes asymptotic with the γ values. Consequently, our system provides useful solutions within a few configurations. This means that our system saves users from examining on extremely large number of possible configurations, within which they probably cannot easily find the desired configurations.

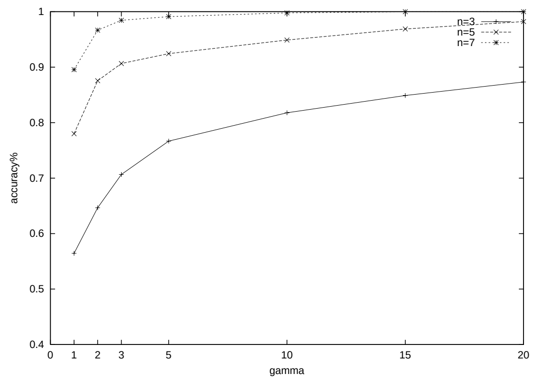


Figure 6: Evaluation of ranking method, where $\varepsilon = 1$.

Alternative Products

In the examination on presenting alternatives, we used another initialization function, but the update function is the same one that uses the navigation function. The initialization function first instantiates a new query clearing all feature values with unknown values. Then, feature `CPU` is specified with the value in the current test case, `c`, while feature `Body Type` is intentionally defined as an incompatible value that cannot hold with the `CPU` value at the same time.

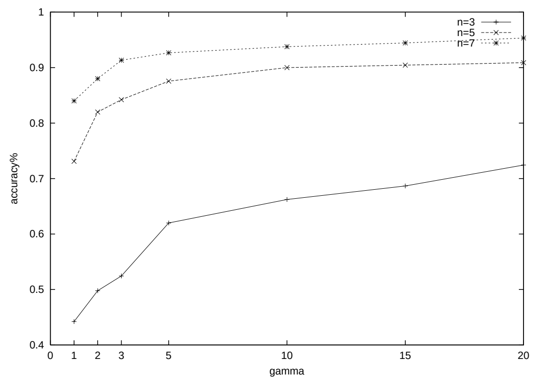


Figure 7: Evaluation of presenting alternative products, where $\varepsilon = 1$.

Figure 7 shows that our system achieves high accuracy and that the accuracy becomes asymptotic at around $\gamma = 5$. In addition, there is a significant difference in accuracy between the examinations with $n = 3$ and $n = 5$. This implies that it becomes more possible to recover from an incompatibility if extra one or two query items are specified. Consequently, our system provides users with opportunities to obtain the desired configurations even when they provide incompatible conditions in their queries. And this opportunity becomes greater, if the users specify a few more query items.

Navigation

In the examination of the navigation function, we employed the initialization function used in examining the ranking method. We employed two versions of update functions for comparison purposes. One has the navigation function; we used the same update function employed in the two examinations above. The other does not have the navigation function, meaning that the next feature to be specified is determined randomly.

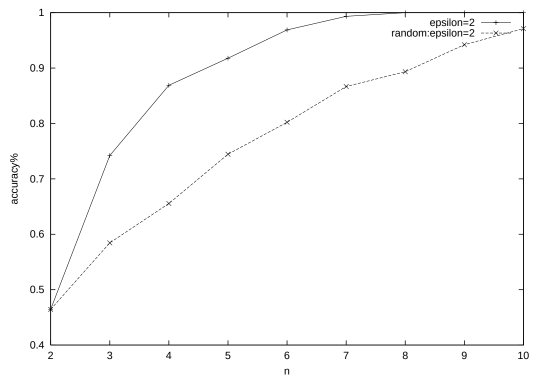


Figure 8: Evaluation of navigation, where $\gamma = 1$.

In Figure 8, the accuracy quickly goes up in the examination with the navigation function, compared to the examination without navigation, where its accuracy increases almost linearly. Consequently, this demonstrates that our system function leads users to the desired configurations with much fewer interactions.

Conclusion

We have proposed a new framework of product configuration that integrates CSP with CBR. By using similar cases to a current query, CBR dynamically generates a description of desirability. The configuration model is composed of this desirability description and the given definition of a product family. Thus, our system obtains the configuration models adapted to individual user queries by using CBR techniques. CSP solver processes this configuration model as a constraint satisfaction problem with ranking the solutions by the object function. As a result, the solutions are arranged in sequence by desirability for a user, and they are correct as well in that they satisfy both the user query and the definition of a product family. Furthermore, our framework presents alternative configurations by relaxing the values of query items. Alternatives are helpful to users who consequently have an opportunity of finding useful configurations and to keep their inquiries active. We applied this framework to an on-line sales system and described its concrete implementation. Our system was evaluated empirically by the accuracy that the system achieves in the leave-one-out cross validation. The result of this experiment support our belief that this framework is practical and works well in the on-line sales system.

Further enhancements of our system is our future work. One enhancement is that our system should be capable of handling revisions of product family. It is not until new products and parts become popular and in circulation that they appear in similar cases to a user query. Accordingly, it is unlikely that these products and parts are ranked higher in solutions even if they are appropriate to some user queries. One implementation to handle this problem is to define the new products as synonyms of their old counterparts so that the revised products could have preference similar to that of the old ones. Another enhancement is that our system should be capable of taking care of the intent of the sales side, such as special sales of a certain product. We will utilize sample cases with which the administrator of a sales system declares a rule of product recommendation, sometimes, with the conditions on user profiles. In addition, scalability is a critical issue in on-line sales system. To handle a huge number of cases, the implementation of storage reduction and case prototyping is expected.

Another direction of our future work is applying CBR Wrapper to other domains. CBR Wrapper may be located between human users and an existing system specific to an application area, such as a design of life insurance and trip planning.

References

Barker, V. E., and O'Connor, D. E. 1989. Expert systems for configuration at Digital: XCON and beyond. *Communication of ACM* 32(3):298–381.

Cost, S., and Salzberg, S. 1993. A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning* 10(1):57–78.

Doyle, M., and Cunningham, P. 2000. A dynamic approach to reducing dialog in on-line decision guides. In *Fifth European Workshop on Case-Based Reasoning*, 49–60. Springer-Verlog.

Fleischanderl, G.; Friedrich, G. E.; Haselböck, A.; Schreiner, H.; and Stumptner, M. 1998. Configuring large systems using generative constraint satisfaction. *IEEE Intelligent Systems* 13(4):59–68.

Gelle, E., and Weigel, R. 1996. Interactive configuration using constraint satisfaction techniques. In *2nd International Conference on Practical Application of Constraint Technology*, 57–82.

Mittal, S., and Fraymann, F. 1989. Towards a generic model of configuration tasks. In *11th International Joint Conference on Artificial Intelligence*, 1395–1401.

Okamoto, S., and Yugami, N. 1997. Theoretical analysis of case retrieval method based on neignborhood of a new problem. In *Second International Conference on Case-Based Reasoning*, 349–358. Springer-Verlag.

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc.

Rahmer, J., and Voss, A. 1996. Case-based reasnoing in the configuration of telecooperation systems. In *Tech. Report FS-96-03*, 93–98. AAAI Press.

Sabin, D., and Weigel, R. 1998. Product configuration frameworks – a survey. *IEEE Intelligent Systems* 13(4):32–85.

Stahl, A., and Bergmann, R. 2000. Applying recursive CBR for the customization of structured products in an electronic shop. In *8th German Workshop on Case-Based Reasoning*.

Stumptner, M., and Wotawa, F. 1998. Model-based reconfiguration. In *Proccedings Artificial Intelligence in Design (AID-98)*.

Tsang, E. 1993. *Foundations of Constraint Satisfaction*. Harcourt Brace & Company.